A REST Service for Triangulation of Point Sets Using Oriented Matroids

José Antonio Valero Medina and Ivan Lizarazo Salcedo

Francisco José de Caldas District University, Colombia

***Abstract* — This paper describes the implementation of a prototype REST service for triangulation of point sets collected by mobile GPS receivers. The first objective of this paper is to test functionalities of an application, which exploits mobile devices' capabilities to get data associated with their spatial location. A triangulation of a set of points provides a mechanism through which it is possible to produce an accurate representation of spatial data. Such triangulation may be used for representing surfaces by Triangulated Irregular Networks (TINs), and for decomposing complex two-dimensional spatial objects into simpler geometries. The second objective of this paper is to promote the use of oriented matroids for finding alternative solutions to spatial data processing and analysis tasks. This study focused on the particular case of the calculation of triangulations based on oriented matroids. The prototype described in this paper used a wrapper to integrate and expose several tools previously implemented in C++.**

***Keywords* — Mobile devices, oriented matroids, spatial data representation, services, triangulation.**

I. Introduction

IN the context of spatial data representation, triangulation of a set of points is essential for many applications. A typical example is the generation of triangulated irregular networks (TINs) to represent ground topography. Another example is the representation of spatial objects using *simplicial complexes* which allows breaking complex objects into simpler underlying geometries. In the case of complex objects in the two-dimensional Euclidean space, simple geometries such as triangles provide a close approximation of spatial objects.

Whatever be the purpose of triangulation, a computational solution is necessary to perform an optimal calculation. Traditional algorithms are based on strategies that rely on geometric representation both of the point set and the Euclidean space embedding it. That is the case of greedy algorithms ([1]) which build triangles using an iterative process based on three criteria: (i) selecting the minimum distance between pairs of points, (ii) not revisiting diagonals that have already been visited, and (iii) testing intersection of new diagonals with those previously obtained. This algorithm has $O(n^3)$ time complexity.

A proposal for simple polygons [2] breaks down the input polygon into monotonous chains to subsequently perform triangulation of each sub-polygon, thus obtaining $O(n)$ time complexity. Another proposal initially finds the Voronoi diagram [3] associated with the set of points, and then calculates the Delaunay triangulation. All of the above cited proposals use traditional geometric representations leading to algorithms suffering from a significant computational complexity. However, in recent decades, there has been an interest for using alternative mechanisms to improve computational efficiency for spatial data processing. [4] has studied the particular case of triangulation of a set of points based on oriented matroids, who implemented several tools using the C++ language.

This article describes the implementation of a prototype REST service for triangulation of a set of points obtained by global positioning system (GPS) receivers. This application targets mobile devices such as tablets and cell phones, using Android OS. The prototype wraps components generated by [4] with the purpose of testing oriented matroids as a means to solve problems in which spatial location matters.

The next section of this paper introduces basic concepts on matroids and oriented matroids as well as the proposal of [4] for triangulation of a point set using only purely combinatorial oriented matroids. The third section describes REST services based on entities that are stored in a persistent repository. This section also makes a description of the application development architecture for Android environment, based on the Eclipse IDE.

The fourth section discusses the architecture of the system and its scheme of access and operation. The fifth section makes a presentation of an overview of the used tools. The sixth section presents a prototype test and the last section formulates conclusions and final remarks.

II. From the Geometric to the Combinatorial

This section introduces basic concepts of oriented matroids. For a more comprehensive treatment, interested readers are referred to [5].

A. Matroid

The concept of matroid was introduced by [6] in the article

DOI: 10.9781/ijimai.2014.264

entitled "On the abstract properties of linear dependence". Whitney describes the approach with respect to a given matrix and its columns, in such a way that any subset of these columns forms a matrix with a particular range. Considering the columns as abstract elements, a matroid with range given by the number of linearly independent columns is formed.

According to [**¡Error! No se encuentra el origen de la referencia.**] from a matrix over any field, it is possible to define a matroid. In particular, the interest from a computational point of view is on finite fields, such as the Galois fields $GF(p^k)$, which give exactly p^k elements when p is a prime number. With $k = 1$ the $GF(p)$ field can be seen as the set $\{0,1,2,\ldots,p-1\}$ with operations of addition and multiplication module *p*. The obtained matroid is called a *vector matroid*.

The concept of graphic matroid is introduced in [7] based on the set of edges of a graph and the set of subsets of arcs of the same graph that does not contain all the arcs in any cycle, with the peculiarity that non isomorphic graphs may have isomorphic matroids.

The formulation of affine matroid [8] is based on the concept of affine independence against affine sets (in $\mathbb{R}^2$ they are the empty set, the points themselves, the straight lines and the plane itself); thus, given a set of points in the plane and a set of affine independent subsets of them, there is a matroid called *affine matroid*.

B. Oriented Matroids

According to [8], geometry of matroids is based on what affine or linear sets provide but it lacks a structure of convexity. There is neither notion of duality or intermediation between the points of a straight line, nor existence of hyperplanes separating the space into two half-spaces. According to [8], vector spaces do not have enough structure to support a theory of convexity; therefore, it is not sound to expect matroids to do it. Authors state the need of providing an additional framing (i.e. an orientation) to the matroids from which the convexity may arise. In a nutshell, that is the theory of oriented matroids.

In [9], it is contended that the oriented matroid of a finite set of points draws information from the relative position and orientation from the configuration, which can be provided by a list of signs that encode the orientation of all its bases. Although in the passage from a specific point's configuration to its oriented matroid, metric information is lost, many of the structural properties have their counterpart in the combinatorial level of oriented matroid. That is to say that oriented matroids describe the structure of incidence between the elements of the matroid and their respective hyperplanes, as well as they encode the position of the elements relative to the hyperplanes; i.e. which items fall on the positive side, which ones on the negative side and which ones inside the hyperplane [10].

C. Computational calculation of triangulations

In [4], it is introduced a procedure to perform triangulation from a set of points based only on the combinatorial structures of the associated oriented matroid. Authors state that triangulations are a basic means for decomposing complex objects into simpler ones. According to [4], for a configuration $\mathcal{A}$ with n $d-dimentional$ points, a T subset of subsets of $\mathcal{A}$, each one made of $d+1$ elements, is a triangulation if and only if

$$\bigcup_{\sigma \in T} conv\ \sigma = conv\ \mathcal{A} \quad (1)$$

and

$$conv\ \sigma \cap conv\ \sigma' = conv(\sigma \cap \sigma')\ \ \forall \sigma, \sigma' \in T \quad (2)$$

The contribution of [4] is that, instead of using expensive linear programming with exact arithmetic to ensure the two aforementioned conditions, authors use purely combinatorial checks based on the oriented matroid $\mathcal{A}$. To check whether the first condition is met the set of all circuits of $\mathcal{A}$ *is used* and it is verified that for every pair of subsets of $\sigma, \sigma' \in T$ there exists a circuit $(Z^+, Z^-) \in \mathcal{A}, Z^+ \subseteq \sigma\ y\ Z^- \subseteq \sigma'$.

On the other hand, to check purely combinatorial fulfillment of the second condition, the set of all co-circuits of $\mathcal{A}$ is used. There is a co-circuit for each affine hyperplane $(d-1)-dimensional$ extending by subsets of $\mathcal{A}$, which includes all points of $\mathcal{A}$ that are on the positive side ($\mathcal{C}^+$) and all the points of $\mathcal{A}$ that are on the downside ($\mathcal{C}^-$) of the hyperplane. For final checking of the second condition, it is verified that, for each $\sigma \in T$, each cocircuit of $\mathcal{A}$ contained by itself, whose sets $\mathcal{C}^+$ and $\mathcal{C}^-$ are both not empty, must have a $\sigma' \in T$ that also contains itself.

In accordance with [4], as long as circuits of $\mathcal{A}$ determine in a purely combinatorial fashion their co-circuits and vice versa, and since the matroid of $\mathcal{A}$ is defined by anyone of such co-circuits, the number of possible triangulations of $\mathcal{A}$ set depends only on its oriented matroid. The passage from the geometry to the combinatorics associated with $\mathcal{A}$ is established through the concept of its chirotopo [11] [12].

$$\chi: \begin{cases} \binom{\mathcal{A}}{d+1} \to \{+,-,0\} \\ (i_1, i_2, \ldots, i_{d+1}) \mapsto sign\left(\det\left(a_{i_1}, a_{i_2}, \ldots, a_{i_{d+1}}\right)\right) \end{cases} \quad (3)$$

A chirotopo assigns its orientation to each ordered base of $\mathcal{A}$. The circuits of $\mathcal{A}$ can be calculated based on all subsets of $d+2$ elements of $\mathcal{A}$ and its cocircuits, using all its subsets of d elements and the calculation of the associated chirotopos. In [4] proposes several algorithmic solutions for triangulation, ranging from obtaining one of the several possible triangulations to obtaining all possible number of triangulations, by using only combinatorial structures associated with the oriented matroid of the points set. These algorithms were implemented in the package TOPCOM (Triangulations of Point Configurations and Oriented

Matroids) looking for a minimum time complexity and a maximum efficiency [4].

III. TECHNOLOGICAL DEVELOPMENT ENVIRONMENT

This section discusses web development concepts useful for using oriented matroids as alternative means for representation and implementation of computational solutions involving spatial location of data.

The main reason for adopting a Web implementation was to provide high availability for the triangulation functionalities proposed by [4]. In a similar way, the choice of the mobile devices environment with Android support leaned on the idea of having a source of sets of points that would be highly available. As it is well known, using a mobile device such as a GPS enabled tablet or a cell phone is something quite common these days.

A. REST webservices

Possibilities for providing online Web capabilities are very different, including solutions based on APIs such as Sockets, XML-RPC or RMI-based components. This article used Web services REST - REpresentational State Transfer [13], as a platform for implementation that provides transparent access to resources possibly including persistent repository storage.

REST was originally introduced as an architectural style for building systems hypermedia distributed on a large scale. The architectural style REST rests on four principles:

--Identification of resources via URIs which outline a set of resources identifying which elements customers interact with.

--Uniform interface which allows user to manipulate the resources using a predefined set of operations: *create* (PUT), *read* (GET), *update* (POST) and *delete* (DELETE).

--Self-descriptive messages, through which resources are uncoupled from their representation to allow content's access in a variety of formats (XML, JSON, etc.).

--Interactions with state through hyperlinks are enabled causing that resource's interactions be stateless, i.e. request messages are self-contained.

The REST services build on well-established W3C protocols (HTTP, XML, etc); thus, their creation should be fairly simple. Effort to building REST services clients is low since testing can be done using any Web browser.

On the other hand, since messages are contained in the URI, a major constraint arises when the associated data set is large enough to exceed the maximum size to be considered a well formed URI.

REST constrains the interface of a resource to its generic uniform interface with predefined operations, and there is very little to choose in terms of available operations. Therefore, a lot of effort should be put on defining which resources need to be exposed. Furthermore, it is necessary to assess whether the four operations are applicable to each resource exposed. In addition, it is also necessary to establish what is the application's semantics associated to each individual resource.

Since REST services are built directly over the HTTP transport protocol, nothing has to be decided about the communication protocol to use. On the other hand, REST has no preset format to adjust the data with. As this issue can be negotiated, it is possible to use different formats such as XML, JSON, or even the SOAP itself.

B. Development environments for Android

Android is undoubtedly one of the most popular platforms for mobile devices in the world such that the manufacturers of this devices use it in a very high proportion. Android is an operating system based on Linux, an open, free and cross-platform operating system, so there are a very nice set of tools for application development. Android provides all the necessary interfaces for developing applications that require access to the basic functionality of mobile devices (i.e. network interface, GPS, etc.) in the Java programming language.

All of the functionalities required in the development of an application for Android are available in its Software Development Kit (SDK). As alternative it is possible to use libraries released by Google as part of its Google Play Services.

The primary IDE environment is the open source tool Eclipse, which also has a plugin, which provides an Android image, with which it is possible to perform application development for Android on any operating system using this emulator for testing [14].

Each Android application uses an own process identified by its ID, and it is the only process accessing user files. The devices have a unique focus, the main application, which is the application visible on the screen, but they can run several applications in the background, each one with its own stack of tasks. The pile of tasks is the sequence of execution of processes in Android. They consist of activities that are stacking as they are invoked, and can only be terminated when the tasks above are completed.

IV. WEB SERVICE FOR TRIANGULATION OF SETS OF POINTS

As mentioned above, the prototype service of point set triangulation was planned in order to be a highly available service, so it was arranged as a Web service REST with the access mode shown in **Fig. 1**.

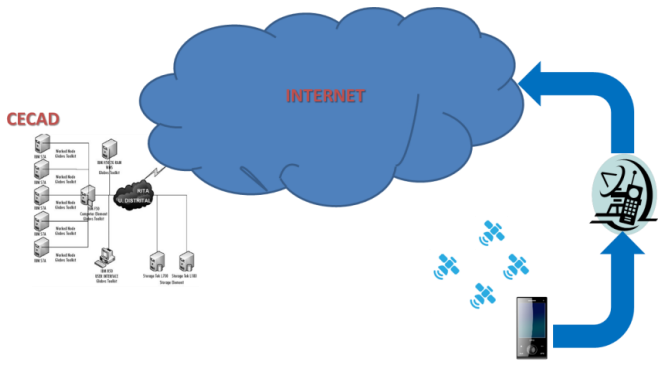

Fig. 1 Access mode of triangulation service based on oriented matroids.

The REST web service was deployed on GlassFish in the Center for High Performance Computing (CECAD) at *Francisco José de Caldas District University*. The service can be accessed from any mobile device connected to the internet via a provider of cellular mobile data, or discovered through a wireless network. The mobile device should have a good coverage of GPS satellites to guarantee a minimum level of accuracy and availability of geographical coordinates.

Fig. 2 shows the components architecture diagram of the system following [15]. For a set of points obtained from the mobile device, it was created a REST server with two REST services from Entity Classes based on the repository defined on the service provided by Java Persistence. *ConjuntodepuntosFacadeREST* is a REST service responsible for generating a group ID of points per session triggered by a connected mobile device with the *read* operation (GET) enabled. This operation returns a list of points formed by a single point with identifier and coordinates dummy but with the new group ID. The reading based on identifier operation (GET {id}) for this REST service was also adapted to allow any connected client to obtain the associated set of points saved to the repository within a given time interval.

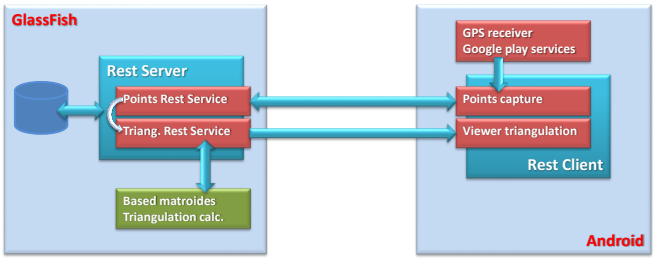

Fig. 2 Triangulation service based on oriented matroids architecture diagram.

TriangulacionesFacadeREST is a REST service created to get the possible triangulations from the current set of points. For this REST service, it was adapted the reading operation based on identifier (GET {id}). Thus, a connected client can request all possible triangulations at any time. For the calculation of the triangles, the REST service was supplemented with a thread responsible for carrying out the wrapper of the *points2triang* component of the software implemented by [4].

The legacy *points2triang* component receives the set of points by its standard input and delivers the calculated triangulations (if it is possible to find any) by its standard output. In order to enable the communication between this component and *TriangulacionesFacadeREST* service, the standard input and output of the component was respectively redirected from and to two auxiliary files. The former is created by the REST service with the coordinate transformation (from the geographic system coordinates to Transversal Mercator) of the points set that the mobile client provide. The latter is generated by the legacy component and used by feed backing the *TriangulacionesFacadeREST*.

Finally, *TriangulacionesFacadeREST* returns back to the mobile client a set of edges for the different possible triangulations associated with the set of points in a session with a client.

The client was implemented based on asynchronous tasks in order to reduce the complexity of the main thread and prevent application crashes.

As mentioned above, the client was developed for the Android system using the API provided by Google Play Services to get coordinates obtained from the receiving device's GPS.

V. TOOLS

This section describes tools used for implementation of the prototype.

A. Building of the web service

The Netbeans IDE was used for the construction of the web service (WS-Rest), following a six-step procedure:

--Creation of tables *ConjuntoDePuntos* and *Triangulaciones*.

--Creation of a new project and addition of connection drivers to the database in Derby.

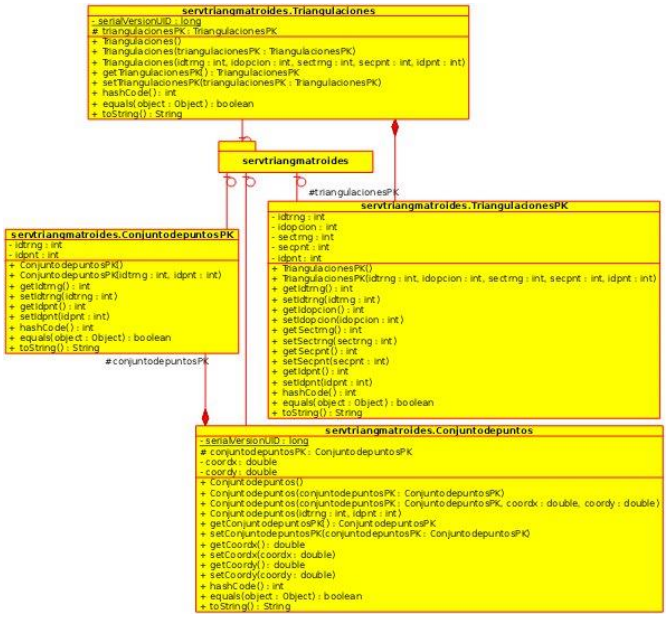

Fig. 3 Entity classes class diagram.

--Creation of a resource (persistent entity) for each table in the database (**Fig. 3**).

--Creation of *ConjuntodepuntosFacadeREST* and *TriangulacionesFacadeREST* REST Web services for each Entity Class created in the previous step (**Fig. 4**).

--Creation of methods that implement the REST operations that will be used in each REST service created. The *ConjuntodepuntosFacadeREST* class is adapted using the *find* (GET {id}) and *findAll* (GET) methods. The *TriangulacionesFacadeREST* class is adapted using the *find* (GET {id}) method.

--Creation of the *Trng* class for the Wrapper of the legacy component responsible for carrying out the triangulation based on the oriented matroid associated to the set of points.

The REST services were available at a public URL allocated by the CECAD of the District University.

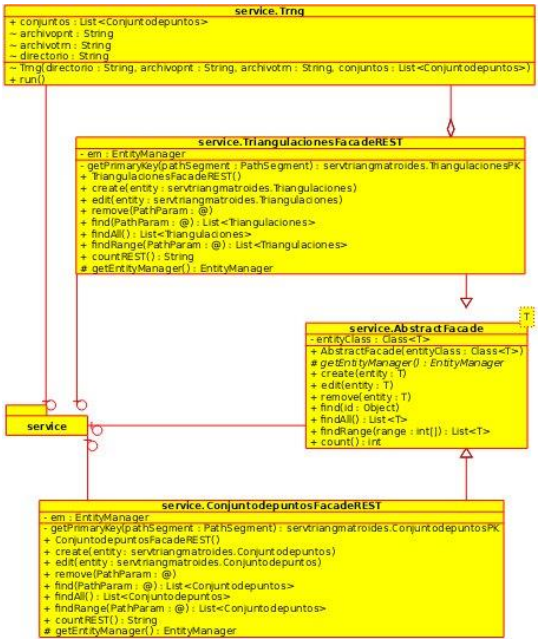

Fig. 4 Services class diagram.

B. Building of the Android (WS-Res) client

For the construction of the Android client providing access to the web service (WS-Rest), following tools were used: Eclipse IDE, the Android Development Tools (ADT) plugin for Eclipse and the Android SDK. Steps for the installation of the mentioned tools were as follows:

--Installing the Eclipse IDE, downloading it from the URL *http://www.eclipse.org/downloads/*. The installation is doing by simple unzip of the downloaded file and run the eclipse program from the eclipse folder.

--Installing the SDK Android available from the URL *http://developer.android.com/sdk/index.html*. After downloading the Android SDK, the installer was run, specifying a path to the JDK. Then, it was needed, through the SDK Manager, to install the "*Android SDK Platform-tools*", "*Android 4.3 (18 API)*" and "*Android 2.2 (8 API)*" platform, and extra "*Android Support Library*" package.

--It was also necessary to create an environment for Android emulation, using the AVD (*Android Virtual Device*) manager of the Android SDK. This emulated environment is useful for test development for Android without using an actual mobile device.

--To complete the tools installation it was necessary to install the *Plugin Android Development Tools* (ADT) for Eclipse using the option "*Install New Software of Eclipse*", providing the URL *https://dl-ssl.google.com/android/eclipse/*, and selecting the two *Developer Tools* and *NDK Plugins* packages.

--Once the software tools were installed, the Android application project in Eclipse and the layout of the GUI client were created.

To support all of the required functionality, it was necessary to make the main activity (*MainActivity*) to extend the *FragmentActivity* class and to implement *ConnectionCallbacks* and *OnConnectionFailedListener* interfaces of the *GooglePlayServicesClient* package, as well as the *LocationListener* interface of the package *com.google.android.gms.location* (**Fig. 5**).

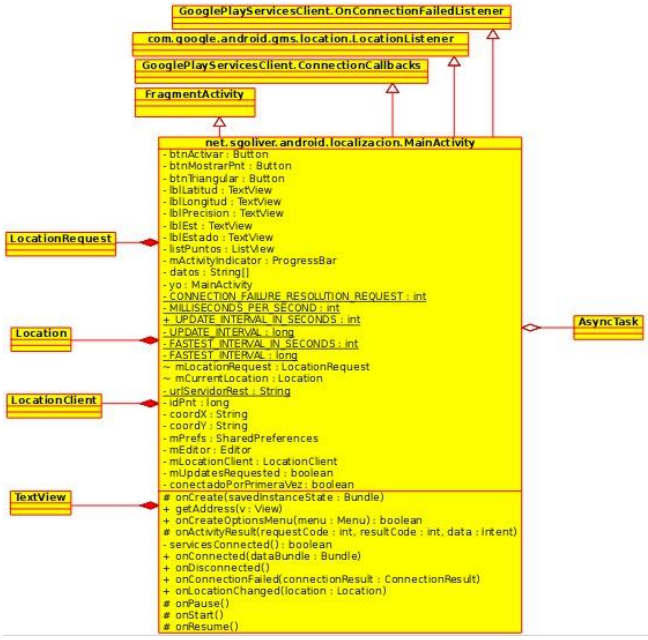

Fig. 5 Client class diagram.

VI. PROTOTYPE TEST

For interaction with the points set triangulation REST service, four asynchronous tasks were defined.

-- A task to obtain the ID of the points set associated with the session initiated by the client; this task starts from the *onCreate* method of the main activity class (**Fig. 6**).

Fig. 6 Triangulation ID assigned on client interface starting.

--When an *onLocationChanged* event is thrown by the Google Play Services software component, a popup message is put on the user interface whit the geographic coordinates provided by the GPS device and the client component put them in the Posición Actual controls zone ().

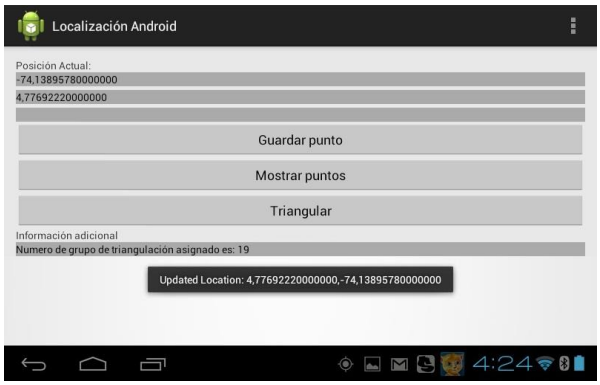

Fig. 7 Detection of Updated Location events on the user interface.

--A second task for the insertion of the point selected by the user (through the action of the "*Guardar punto*" button on the user interface) provided by the GPS receiver and displayed in the user interface in the field "*Posición actual*" (**Fig. 8**).

Fig. 8 Storage by the service using the user interface.

--A third task for obtaining the set of points stored so far from the REST service (through the action of the "*Muestre puntos*" button of the user interface) (**Fig. 9**).

Fig. 9 Stored Points set provided by the service.

--Finally, a task to obtain the edges of the possible triangulations reachable from the saved set of points. Both the list of points and the edges of the possible triangulations are presented in the *ListView* that is shown at the bottom of the user interface.

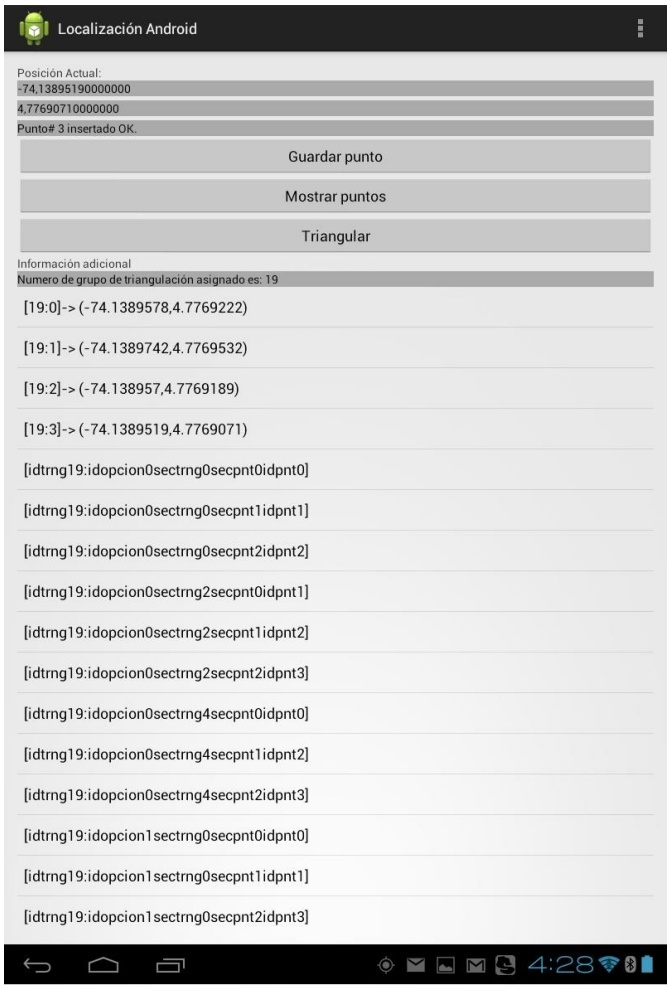

Fig. 10 Stored points set obtained from the service and its possible triangulations.

Fig. 10 shows the user interface with four points provided by the mobile device (a Toshiba Excite 10 inches tablet with a wireless link) and their possible triangulations. For each possible triangulation (identified by *idopcion*), the triangle is shown (identified by *sectrn*). For each triangle the point sequence (indicated by *secpnt*) and the point's id (identified by *idpnt*) is shown. The test used four very near points only a few meters apart and in this case, the particular point's arrangement produced the only two possible triangulations in agreement with the based matroid combinatorial algorithm used.

Coordinate transformation from the original World Geodetic System (WGS-84) spatial reference system into the Transverse Mercator projection was conducted using the minimum latitude and longitude of the set of points as the *latitude of origin* and *central meridian* parameters. This spatial

coordinate's projection is needed as GPS receivers provide geographic coordinates.

The test was also conducted using a LG Optimus L7 Cell smartphone. The following explanation focuses on illustrating the REST's server side of this test. **TABLE I** shows the 12 points provided by the smartphone. These points were located at a farther distance than the points in the previous test.

TABLE I
POINT SET LIST ON SERVICE SIDE

Geographic coordinates JSON format
World Geodetic System (WGS-84)


```
[{"conjuntodepuntosPK":{"idtrng":23,"idpnt":0},
    "coordx":-74.1465482,"coordy":4.7710368},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":1},
    "coordx":-74.1429136,"coordy":4.7686211},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":2},
    "coordx":-74.141702,"coordy":4.7678158},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":3},
    "coordx":-74.13895,"coordy":4.7768594},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":4},
    "coordx":-74.1328556,"coordy":4.7716231},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":5},
    "coordx":-74.1354637,"coordy":4.7729735},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":6},
    "coordx":-74.1429136,"coordy":4.7686211},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":7},
    "coordx":-74.1366037,"coordy":4.7763712},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":8},
    "coordx":-74.1365644,"coordy":4.7764313},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":9},
    "coordx":-74.1465482,"coordy":4.7710368},
 {"conjuntodepuntosPK":{"idtrng":23,"idpnt":10},
    "coordx":-74.140837,"coordy":4.7762281},
 {"conjuntodepuntosPK": {"idtrng":23,"idpnt":11},
    "coordx":-74.1409072,"coordy":4.7760984}]
```


The point primary key *conjuntodepuntosPK* is composed of the triangulation id *idtrng* and the point id *idpnt*. *coordx* and *coordy* are the point´s meridian and latitude respectively.

When the client mobile device asks to performing the triangulation, the original coordinates are projected as shown in **TABLE II**, using for this test the values *4.7678158* and *-74.1465482* (the minimums) as the *latitude of origin* and *central meridian* parameters respectively.

TABLE II
PROJECTED POINT SET LIST

Id	X	Y
0	3561	0
1	890	4032
2	0	5376
3	10000	8429
4	4210	15190
5	5703	12296
6	890	4032
7	9460	11031
8	9527	11075
9	3561	0
10	9302	6335
11	9159	6257

The projected coordinates were scaled by a 10 factor.

TABLE III shows the legacy component result returned to the triangulation service REST.

TABLE III
THE LEGACY COMPONENT RESULT


```
[T[0]:=[0->12,3:{{0,1,2},{0,2,3},{2,3,4},{3,4,7},{4,7,8},{3,7,8},{0,3,10}}];]
[T[1]:=[1->12,3:{{0,1,2},{0,2,3},{2,3,4},{3,4,7},{4,7,8},{3,7,8},{3,10,11},
    {0,10,11},{0,3,11}}];]
[T[2]:=[2->12,3:{{0,1,2},{0,2,3},{4,7,8},{3,7,8},{0,3,10},{2,4,7},{2,3,7}}];]
[T[3]:=[3->12,3:{{0,2,3},{2,3,4},{3,4,7},{4,7,8},{3,7,8},{0,3,10},{0,2,6}}];]
[T[4]:=[4->12,3:{{2,3,4},{3,4,7},{4,7,8},{3,7,8},{0,3,10},{1,2,3},{0,1,3}}];]
[T[5]:=[5->12,3:{{0,1,2},{3,4,7},{4,7,8},{3,7,8},{0,3,10},{0,3,4},{0,2,4}}];]
[T[6]:=[6->12,3:{{0,1,2},{0,2,3},{2,3,4},{0,3,10},{3,4,8}}];]
[T[7]:=[7->12,3:{{2,3,4},{3,4,7},{4,7,8},{3,7,8},{1,2,9},{2,3,9},{3,9,10}}];]
[T[8]:=[8->12,3:{{0,1,2},{0,2,3},{3,4,7},{4,7,8},{3,7,8},{0,3,10},{3,4,5},{2,4,5},
    {2,3,5}}];]
[T[9]:=[9->12,3:{{0,1,2},{2,3,4},{3,4,7},{4,7,8},{3,7,8},{2,3,10},{0,2,10}}];]

…
[T[514]:=[514->12,3:{{0,1,2},{0,2,10},{2,4,8},{3,8,10},{2,8,10}}];]
[T[515]:=[515->12,3:{{0,1,2},{2,4,5},{2,3,10},{0,2,10},{2,3,8},{4,5,8},{2,5,8}}];]
[T[516]:=[516->12,3:{{4,7,8},{3,7,8},{2,4,7},{3,9,10},{1,3,9},{1,3,7},{1,2,7}}];]
[T[517]:=[517->12,3:{{3,4,7},{4,7,8},{3,7,8},{3,10,11},{0,10,11},{1,2,3},{3,4,5},
    {2,4,5},{2,3,5},{0,1,11},{1,3,11}}];]
[T[518]:=[518->12,3:{{3,4,7},{4,7,8},{3,7,8},{3,10,11},{1,2,3},{3,4,5},{2,4,5},
    {2,3,5},{1,3,9},{9,10,11},{3,9,11}}];]
[T[519]:=[519->12,3:{{3,4,7},{4,7,8},{3,7,8},{3,10,11},{0,10,11},{0,3,11},
    {0,1,3},{3,4,5},{2,4,5},{1,3,5},{1,2,5}}];]
[T[520]:=[520->12,3:{{3,4,7},{4,7,8},{3,7,8},{3,10,11},{0,10,11},{0,3,11},
    {0,2,6},{3,4,5},{2,4,5},{0,3,5},{0,2,5}}];]
[T[521]:=[521->12,3:{{0,1,2},{3,7,8},{0,3,10},{0,2,4},{3,5,7},{0,3,5},{0,4,5},
    {4,5,8},{5,7,8}}];]
```


The point primary key *conjuntodepuntosPK* is composed of the triangulation id *idtrng* and the point id *idpnt*. *coordx* and *coordy* are the point´s meridian and latitude respectively.

In this case, the particular point's arrangement produced 521 triangulations (all the possible ones). TABLE III shows only the result's head and tail. Each possible triangulation takes no more than the point id's as the process uses only the point set combinatorial structure.

Finally, the four last triangles of the last possible

triangulation of the response to the client mobile device are shown in **TABLE IV** using the JSON format.

TABLE IV
TRIANGULATIONS LIST ON SERVICE SIDE

…

{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":10,"secpnt":0,"idpnt":0}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":10,"secpnt":1,"idpnt":3}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":10,"secpnt":2,"idpnt":5}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":12,"secpnt":0,"idpnt":0}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":12,"secpnt":1,"idpnt":4}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":12,"secpnt":2,"idpnt":5}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":14,"secpnt":0,"idpnt":4}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":14,"secpnt":1,"idpnt":5}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":14,"secpnt":2,"idpnt":8}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":16,"secpnt":0,"idpnt":5}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":16,"secpnt":1,"idpnt":7}},
{"triangulacionesPK":{"idtrng":23,"idopcion":521,"sectrng":16,"secpnt":2,"idpnt":8}}]

Each possible triangulation (identified by *idopcion*) shows the triangle (identified by *sectrn*), the point sequence (indicated by *secpnt*), and the point's id (identified by *idpnt*).

VII. CONCLUSION

The implementation of the prototype presented here allowed the reuse of a legacy tool implemented in C++ for triangulation of point configurations in two-dimensional spaces (for a discussion of performance details see [4]). The inherited tool was integrated in a Rest Web service using the point settings obtained from an Android mobile device.

Android client's interaction for coordinates' collection and use of the triangulation service was greatly simplified by the adoption of REST services. However, adaptation of the semantics of the operations GET and PUT, did not turn out as "natural" as it might be expected.

As definition of the services REST, for both the sets of points and the possible triangulations, was based on the Entity Class associated with the persistence provided by Java Persistence Service, the interchange format output by the triangulation component was not as self-descriptive as the one supplied by the service.

The projected coordinates were scaled by a ten factor in order to have integer values (as required by *points2triang*, the legacy component) but having a decimeter precision. The introduction of this scale factor did not affect in any way the outcome because the procedure used by the legacy component takes in account only the combinatorial structure of the associated oriented matroid.

The REST service asynchronous interaction facilitated the client implementation faults' tolerance, mainly the GPS receiver disconnection one, being necessary only to define the state variables with REST service session as static due that Google Play Services destroys and creates newly its objects after each disconnection.

No statistical validation was done because that was beyond the scope of the work, but this could be the subject of future work.

The matroid-based triangulation service can be reached at the IP address 200.69.103.29 on the http port 22095. The Android Client component (UbicacionGeografica.apk) and the other resources (i.e., IDE projects for Netbeans and eclipse-kepler for the services and client respectively) are available at IDE projects temporary site.

ACKNOWLEDGMENTS

We would like to thank the Francisco José de Caldas District University's Center for High Performance (CECAD) for their prompt and ongoing support, and to providing the computational resources required for the implementation of the prototype and the allocation of a public IP address to access the REST services.

REFERENCES

[1] Worboys, M. and Duckham, M. (2004) GIS: A Computing Perspective, 2Nd Edition, CRC Press, Inc., Boca Raton, FL, USA.

[2] Garey, M. R., Johnson, D. S., Preparata, F. P., and Tarjan, R. E. (1978) Information Processing Letters 7(4), 175 – 179.

[3] deBerg, M., Cheong, O., vanKreveld, M., and Overmars, M. (2008) Computational Geometry, Springer Berlin Heidelberg, third edition.

[4] Pfeifle, J. and Rambau, J. (2003) Computing triangulations using oriented matroids In Michael Joswig and Nobuki Takayama, (ed.), Algebra, Geometry and Software Systems, pp. 49–75 Springer Berlin Heidelberg.

[5] Valero, J. A. (2013) UD y la Geomática 0(7).

[6] Whitney, H. (1935) American Journal of Mathematics 57, 509–533. Oxley, J. (2007).

[8] Stell, J. and Webster, J. (2007) Computers, Environment and Urban Systems 31(4), 379 – 392 Topology and Spatial Databases.

[9] Richter-Gebert, J. and Ziegler, G. M. (2004) In Jacob E. Goodman and Joseph O'Rourke, (ed.), Handbook of Discrete and Computational Geometry, Boca Raton, FL, USA: CRC Press, Inc.. pp. 129–151.

[10] Anderson, L. and Delucchi, E. (2012) Discrete & Computational Geometry 48(4), 807–846.

[11] De Loera, J. A., Rambau, J., and Santos, F. (2010) Triangulations: Structures for Algorithms and Applications, Springer Publishing Company, Incorporated, 1st edition.

[12] Tsukamoto, Y. (2013) Discrete & Computational Geometry 49(2), 287–295.

[13] Pautasso, C., Zimmermann, O., and Leymann, F. (2008) In Proceedings of the 17th International Conference on World Wide Web WWW '08 New York, NY, USA: ACM. pp. 805–814.

[14] Manuel Báez, Álvaro Borrego, J. C., Cruz, L., González, M., Hernández, F., Palomero, D., deLleraand Daniel Sanz, J. R., Saucedo, M., Torralbo, P., and ÁlvaroZapata (2010) Introducción a Android, E.M.E. Editorial ©.

[15] Kovachev, D. and Klamma, Framework for Computation Offloading in Mobile Cloud Computing, R. 12/2012 2012 International Jorunal of Interactive Multimedia and Artificial Intelligence 1(7), 6–15. DOI: 10.9781/ijimai.2012.171

José Antonio Valero Medina is professor at the Facultad de Ingeniería at Francisco José de Caldas District University in Bogotá Colombia. He holds Msc degree in Teleinformática from Francisco José de Caldas District University. Actually he is a student of the Engineer PhD program there. His main areas of research are distributed systems and Geoinformatics.

Ivan Lizarazo Salcedo is professor at the Engineer Faculty at Francisco José de Caldas District University in Bogotá Colombia. He holds Master in Geographic Information Science and PhD In Geography degrees from University Of London. His main areas of research are Geographic-Object Based Image Analysis and Integration of agent-based simulation and GIS.